Untyped Lambda-Calculus with Input-Output (Progress Report)

Jerzy Tiuryn*
Institute of Informatics
Warsaw University
Banacha 2
02-097 Warsaw, POLAND
tiuryn@mimuw.edu.pl

Mitchell Wand†
College of Computer Science
Northeastern University
360 Huntington Avenue, 161CN
Boston, MA 02115, USA
wand@ccs.neu.edu
http://www.ccs.neu.edu/home/wand

May 31, 1995

Abstract

We introduce an untyped λ-calculus with input-output, based on Gordon's continuation-passing model of input-output. This calculus is intended to allow the classification of possibly infinite input-output behaviors. We introduce a property, called "losslessness," which is a natural property of any reasonable model of this calculus. We then characterize the largest lossless precongruence that refines simulation for this calculus.

*This work is partly supported by NSF Grants CCR-9417382, CCR-9304144, and by Polish KBN Grant 2 P301 031 06.

†Work supported by the National Science Foundation under grants number CCR-9304144 and CCR-9404646.

1 Introduction

The usual notions of contextual equivalence are a generally satisfactory formalization of experiments on the finite behavior of programs: to see whether a program M has some finite behavior, one can easily set up a context $C[-]$ such that $C[M]$ terminates iff M has the desired behavior. If, however, we wish to classify programs that have potentially infinite behaviors, such as a program that echoes its input, then a more direct model of input-output seems necessary.

Gordon [1] provides four such calculi and gives translations between them. Gordon's system involves an elaborate type system. Our goal is to study the simplest such system: an untyped system. Our system is also an simplification of the system considered in [5].

2 The Basic System

The syntax of our terms is given by the following:

$$M ::= v \mid (MM) \mid \lambda v.M \mid c_0(M) \mid c_1(M) \mid r(M)$$

This is the usual untyped λ-calculus, augmented by new unary term constructors c_0, c_1, and r. Let Λ denote this set of terms, and let Λ_0 denote the set of closed terms. We will write c_0M for $c_0(M)$, etc., where convenient. We will use the usual λ-calculus notions of β-reduction, head-reduction, etc.

We turn the closed terms of this calculus into a labelled transition system with actions $\{!0, !1, ?0, ?1\}$ by the following rules:

$$\frac{M =_\beta c_0(M')}{M \xrightarrow{!0} M'} \qquad \frac{M =_\beta c_1(M')}{M \xrightarrow{!1} M'}$$

$$\frac{M =_\beta r(M')}{M \xrightarrow{?0} (M'\ulcorner 0 \urcorner)} \qquad \frac{M =_\beta r(M')}{M \xrightarrow{?1} (M'\ulcorner 1 \urcorner)}$$

where $\ulcorner 0 \urcorner$ and $\ulcorner 1 \urcorner$ are the Church numerals for 0 and 1.

With these rules, we can see that c_0 and c_1 play the role of "print 0" and "print 1", and r plays the role of "read a bit". Note that these terms operate only at the top level, so they are effectively in continuation-passing style. Note that a term with no head-normal form, or a term with some head-normal form other than these four, such as $\lambda x.x$, deadlocks.

For example, the term

$$Y(\lambda f.r(\lambda x.\,\text{zero?}\,x(c_0 f)(c_1 f))) \tag{1}$$

where Y is the usual fixpoint combinator and zero? is a zero-testing term, echoes its input. For a more complex example, here is a term that echoes its input at half-speed by reading two bits, depositing them in an unbounded queue, and printing one bit from the queue:

$$
\begin{aligned}
&Y(\lambda fq.r(\lambda x.r(\lambda y.\, \mathrm{deq}(\mathrm{enq}(\mathrm{enq}\, qx)y) \\
&\qquad\qquad\qquad (\lambda vq'.\, \mathrm{zero?}\, v(c_0(fq'))(c_1(fq')))))))\, q_0
\end{aligned}
\tag{2}
$$

Here q_0 is a coding of the empty queue, $\mathrm{enq}\, q\, x$ returns the queue obtained by adding x to tail of q, and $\mathrm{deq}\, qf$ passes to f the first element on the queue q and the queue obtained by removing that first element. Such terms are of course easy to code in the untyped λ-calculus.

This calculus inherits the usual notions of simulation and bisimulation from its definition as a labelled transition system [4]. By specializing the general definitions to our system, we can characterize these notions as greatest fixpoints of monotone operators on relations:

Definition 1 *If $R \subseteq \Lambda_0 \times \Lambda_0$ is any relation on closed terms, the relation $[R]_B \subseteq \Lambda_0 \times \Lambda_0$ is defined by $M\ [R]_B\ N$ iff the following properties hold:*

1. *if $M =_\beta c_0 M'$, then there exists N' such that $N =_\beta c_0 N'$ and $M'\ R\ N'$.*
2. *if $M =_\beta c_1 M'$, then there exists N' such that $N =_\beta c_1 N'$ and $M'\ R\ N'$.*
3. *if $M =_\beta rM'$, then there exists N' such that $N =_\beta rN'$ and $(M'\lceil 0 \rceil)\ R\ (N'\lceil 0 \rceil)$ and $(M'\lceil 1 \rceil)\ R\ (N'\lceil 1 \rceil)$*

We can define *simulation* $\leq_B$ in the usual way as the greatest fixpoint of $[-]_B$. Bisimulation is the greatest fixed point of $[-]_B \cap ([-]_B)^{op}$. It is immediate that these definitions agree with the general definitions on our system.

It is straightforward to check that the terms (1) and (2) above are not bisimilar, whereas a term that differed from (2) only by altering the coding of queues would be bisimilar to it, even though it would not be equal to it in any reasonable λ-theory.

3 From Bisimulation to Precongruence

While bisimulation gives a good notion of observable equivalence, it is insufficient as an equational theory of terms, because it is not a congruence. For example, the terms $I = \lambda x.x$ and $\Omega = (\lambda x.xx)(\lambda x.xx)$ are bisimilar, because both deadlock, but $I(c_0(I))$ and $\Omega(c_0(I))$ are not bisimilar, since the former has a !0 action but the latter does not.

Thus our situation differs from the usual situation in concurrency theory, in which one seeks to show that bisimulation is a congruence; our goal is to extend the clauses of the definition of $[-]_B$ to get a smaller operator $[-]$ whose greatest fixpoint is a precongruence (a preorder that respects term constructors).

The examples above suggests that $[-]_B$ is insufficiently strict in the case that the head normal form of M is an abstraction. So we define our new generator $[-]$ by adding to Definition 1 the clause:

4. if $M =_\beta \lambda x.M'$ then for all closed P, $(MP)\ R\ (NP)$.

We define $\leq$ as the greatest fixpoint of $[-]$. The coinduction principle states that if $R \subseteq [R]$, then $R \subseteq\ \leq$.

Note that $\Omega \leq M$ for all M, since none of the conditions hold for Ω. Furthermore $\lambda x.\Omega \leq M$, since for all closed P, $(\lambda x.\Omega)P =_\beta \Omega \leq (MP)$. Thus in our system $\lambda x.\Omega \leq \Omega$ and $\Omega \leq \lambda x.\Omega$. This is as desired, since it can be shown that these terms have the same input-output behavior in any context.

Definition 2 *If $R \subseteq \Lambda_0 \times \Lambda_0$, define $R^o \subset \Lambda \times \Lambda$ by $M\ R^o\ N$ iff $(M\sigma)\ R\ (N\sigma)$ for every closing substitution σ. If $R \subset \Lambda \times \Lambda$, define $R_0 \subset \Lambda_0 \times \Lambda_0$ as $R_0 = R \cap (\Lambda_0 \times \Lambda_0)$.*

The first main result is:

Theorem 1 (Precongruence) *$\leq^o$ is a precongruence.*

Proof: (Sketch). The obvious approach is to use the techniques of Howe [3, 2]. Unfortunately, Howe's techniques require to $[-]$ to have the property that for any R and S, $[R]\circ[S] \subseteq [R \circ S]$. This is false for $[-]$: let $R = \{(IP, (\Omega P)) \mid P$ closed$\}$ (where $I = \lambda x.x$) and let $S = \emptyset$. Then $(I, \Omega) \in [R]$, and $(\Omega, \Omega) \in [S]$, but $(I, \Omega) \notin [R \circ S] = [\emptyset]$.

Instead, we introduce another monotone operator $[-]_*$, such that $\mathrm{gfp}([-]_*) = \mathrm{gfp}([-])$ and $[-]_*$ is a simulation generator in the sense of Howe. We then apply Howe's method to $[-]_*$. ∎

4 Contextual and Lossless Precongruences

Theorem 1 shows that $\leq^o$ is a precongruence contained in $\leq^o_B$. Is it the largest such precongruence? The answer is no: the largest precongruence contained in $\leq^o_B$ is always the contextual precongruence, defined as

$$\{(M, N) \in \Lambda \times \Lambda \mid \forall \text{ contexts } C,\ C[M] \leq_B C[N]\}$$

$I \not\leq \Omega$, so $c_0(I) \not\leq c_0(\Omega)$. Yet it can be shown that for any context C, $C[c_0(I)]$ and $C[c_0(\Omega)]$ are bisimilar.

On the other hand, the largest precongruence may conflate terms that are separated in any reasonable semantics. For example, we would expect that in any reasonable semantics, the meaning of c_0 should be injective, on the grounds that prefixing a process by a write operation shouldn't lose any information about that process. That is, if $[\![-]\!]$ is the map from terms to some domain of meanings, ordered by $\sqsubseteq$, not only should we have

$$[\![M]\!] \sqsubseteq [\![N]\!] \Rightarrow [\![c_0(M)]\!] \sqsubseteq [\![c_0(N)]\!]$$

but also

$$[\![c_0(M)]\!] \sqsubseteq [\![c_0(N)]\!] \Rightarrow [\![M]\!] \sqsubseteq [\![N]\!]$$

Similarly, we expect $[\![rM]\!]$ to lose no information compared with $[\![M\lceil 0\rceil]\!]$ and $[\![N\lceil 1\rceil]\!]$. We can translate this notion back into preorders on terms as follows:

Definition 3 *Let $R \subset \Lambda \times \Lambda$. We say R is* lossless *iff*

1. *if $c_0(M)\ R\ c_0(N)$, then $M\ R\ N$*
2. *if $c_1(M)\ R\ c_1(N)$, then $M\ R\ N$*
3. *if $r(M)\ R\ r(N)$, then $(M\lceil 0\rceil)\ R\ (N\lceil 0\rceil)$ and $(M\lceil 1\rceil)\ R\ (N\lceil 1\rceil)$*

We can then show:

Theorem 2 *$\leq^o$ is the largest lossless precongruence R on Λ such that $R_0 \subseteq \leq_B$.*

References

[1] Andrew D. Gordon. *Functional Programming and Input/Output.* Cambridge University Press, Cambridge, 1994.

[2] Douglas J. Howe. A Note on Proving Congruence of Bisimulation in a Generalized Lambda Calculus. unpublished note.

[3] Douglas J. Howe. Equality in Lazy Computation Systems. In *Proc. 4th IEEE Symposium on Logic in Computer Science*, pages 198–203, 1989.

[4] Robin Milner. Operational and Algebraic Semantics of Concurrent Processes. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 1201–1242. MIT Press/Elsevier, 1990.

[5] Mitchell Wand. Compiler Correctness for Parallel Compilers. In *Functional Programming Languages and Computer Architecture*, June 1995. to appear.